PATENT APPLICATION OF

GAGAN V. GUPTA
34322 Torrey Pine Lane,
Union City, CA   94587
Citizenship: INDIA

and

MENGCHEN YU
1457 Lemos Lane, Fremont, CA   94539
Citizenship: China

ENTITLED

FLOATING POINT DIVIDE AND SQUARE ROOT PROCESSOR

FLOATING POINT DIVIDE AND

SQUARE ROOT PROCESSOR

FIELD OF THE INVENTION

This invention relates to floating point
arithmetic units for a central processing unit (CPU),
and particularly to a floating point divide and square
root processor that requires minimal space in an
integrated circuit (IC).

BACKGROUND OF THE INVENTION

Floating point divide and square root
processors require a considerable physical region of
an IC. To conserve space, divide and square root
functions are usually implemented employing iterative
algorithms so that the same hardware is utilized
iteratively over multiple cycles to produce the final
result. Depending on the particular design, each
iteration of the algorithm produces either a single
bit or multiple bits of the divide or square root
result. Generally, processors that produce multiple
bits each iteration can produce the final result in a
shorter time period than processors that produce
fewer or single bits each iteration. Also generally,
the amount of hardware increases with the number of
bits produced each iteration. Consequently,
processors that produce multiple bits during each
iteration require more IC space than processors that
produce a single bit. Moreover, as the hardware
increases, operation frequency is reduced. Hence, IC
designers try to design divide and square root

processors that strike a compromise between area and speed.

Most divide processors use a Radix-4 algorithm that produces up to two quotient bits during each iteration. However, the Radix-4 algorithm is complex, significantly more complex than a Radix-2 algorithm which produces one quotient bit each iteration. Moreover, the Radix-4 algorithm requires more hardware than the Radix-2 algorithm.

The algorithms that implement divide and square root functions are often similar. To conserve space on the IC, designers try to utilize as much of the divide hardware as possible in the implementation of the square root functions. However, these techniques have not been altogether successful. More recently, designers have implemented the divide function using the multiplier array of the arithmetic unit of the CPU, providing additional support for square root. However, the multiplier array also consumes a large amount of area on the IC, and can be frequency limited, so little is gained with this approach. Accordingly, there is a need for a divide and square root processor with improved spatial requirements.

SUMMARY OF THE INVENTION

In one embodiment, an iterative mantissa calculator calculates a quotient mantissa based on a divisor mantissa and a dividend mantissa or a square-root result mantissa based on an operand mantissa.

The calculator includes at least first and second summing devices and at least first and second selectors. In the divide mode, the summing devices are arranged to calculate respective partial remainders during respective iterations and the selectors are responsive to the partial remainders to select a bit for an accumulated quotient. In preferred embodiments of the divide mode, each summing device calculates a respective estimated partial remainder W[j+1] for the next iteration, j+1, as 2*W[j]-S_{j+1}*D, where W[j] is the estimated partial remainder for the current iteration calculated during the prior iteration, S_{j+1} is the selected quotient bit estimated, and D is the respective divisor bit.

In the square-root mode, the first and second summing devices operate during a single iteration to generate a partial remainder W[j+1] and the second selector selects a result bit S_{j+1}. During each iteration, j, the first summing device calculates 2W[j]-2S[j]S_{j+1}, where W[j] is the partial remainder that was estimated by the second summing device during the prior iteration, S[j] is the accumulated result to the current iteration, and S_{j+1} is the selected result bit. A shift register shifts the value of the accumulated result to generate $-S_{j+1}^2 \bullet 2^{-(j+1)}$. The second summing device adds the shifted result to the results of the first summing device to generate the partial remainder for the square root mode.

In preferred embodiments, result adder is responsive to the partial remainder and the accumulated result or quotient to provide the result or quotient mantissa.

5 <u>BRIEF DESCRIPTION OF THE DRAWINGS</u>

FIG. 1 is a functional block diagram of a floating point divide and square root processor according to one embodiment of the present invention.

FIG. 2 is a functional block diagram of
10 divide/square root calculator used in the processor of FIG. 1.

<u>DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS</u>

The present invention is directed to a divide and square root processor that meets a
15 compromise between area and speed to implement the divide function. The square root function is implemented using the same hardware with minimal addition. Simple hardware, and hence low area and high speed, is achieved by using a SRT Radix-2 type
20 algorithm.

FIG. 1 is a block diagram of the functional unit of a divide and square root processor according to one embodiment of the invention. The processor includes an operand unpack 10 that receives the
25 divisor and dividend for a divide operation via busses 12 and 14, respectively. An exponent calculator 16 receives the divisor exponent and dividend exponent via busses 18 and 20, respectively.

For a divide operation $(\frac{\text{Dividend}}{\text{Divisor}})$, divisor bus 12 carries the divisor and dividend bus 14 carries the dividend. Divisor exponent bus 18 carries the exponent part of the divisor, and dividend exponent 20 carries the exponent part of the dividend. For a square root operation $(\sqrt{\text{operand}})$, dividend bus 14 carries the operand, and dividend exponent bus 20 carries the exponent of the operand. Divisor bus 12 and divisor exponent bus 18 are not used for square root operations.

For a divide operation, the resultant exponent is calculated based on the inputs. The exponent is adjusted, as required, based on the final mantissa result. For a square root operation, computation of the resultant exponent is based solely on the operand (dividend) exponent.

Divide/square root calculator 22 iteratively computes the mantissa results, 2 bits during each cycle for divide operations and 1 bit during each cycle for square root operations, and accumulates the result in a redundant number format.

Calculator 22 employs at least two carry-save adders, arranged in a cascaded manner. In the divide mode, each carry-save adder generates a result bit during a respective iteration, so that during each cycle through the carry-save adders at least two result bits are accumulated in a partial remainder.

In the square root mode, the cascaded carry-save adders generate one result bit during each cyclé.

Each carry-save adder in calculator 22 accumulate a partial remainder based on an SRT Radix-2 algorithm. Upon completion of the process by calculator 22, the partial remainders accumulated through the last cycle are processed by result adder 24.

The SRT algorithm is similar to a traditional division algorithm, where the result is computed one digit at a time. At each step a partial remainder is calculated which is used in the next step. The recurrence equation for division is defined as:

$$W[j+1] = 2*W[j] - S_{j+1}*D,$$

where W[j] is the partial remainder of jth iteration, D is the divisor, S_{j+1} is the quotient bit for the (j+1)th iteration, and W[j+1] is the new partial remainder for the (j+1)th iteration. The SRT division algorithm generates a partial remainder W[j+1] on each iteration of the divide operation, subtracting a multiple of the divisor from the scaled current partial remainder W[j].

The SRT algorithm is implemented in redundant residual form. The carry-save adders compute the partial remainders, with each partial remainder being represented by two numbers, the sum (Ws) and the carry (Wc). The real value of the partial remainder is derived from the sum of Ws and

Wc. The quotient estimate is obtained from the partial remainder.

The square root algorithm computes one result bit during each cycle. At each iteration a
5   partial remainder is calculated which is used in the next iteration. The recurrence equation for square root is defined as:

$$W[j+1] = 2*W[j] - 2S[j]S_{j+1} - S_{j+1}^2 \bullet 2^{-(j+1)} ,$$

where W[j]is the partial remainder estimated for the
10   j-th iteration, S[j] is the accumulated result to the j-th iteration, S_{j+1} is the result bit for the (j+1)th iteration, and W[j+1] is the new partial remainder for the (j+1)th iteration. The square root algorithm generates a partial remainder W[j+1] on each
15   iteration of the square root operation. Unlike the divide algorithm where two result bits are generated during the two iterations of each cycle, the square root algorithm generates a single result bit during each cycle.

20   At each iteration, the next result bits are computed based on the partial remainder of the prior iteration (W[j]). The result is accumulated in calculator 22. After all the requisite result bits have been computed, the accumulated partial remainder
25   is sent to the remainder zero and sign detect portion of block 24. The accumulated result may be adjusted in block 24 based on the input from zero and sign detect functions.

The remainder zero/sign detect and result adder block 24 receives input in redundant format from the divide/square root calculator 22 and generates a final 2's compliment value. When

5 calculator 22 outputs partial remainders, the remainder zero/sign detect portion of block 24. performs a zero detect and also checks for the sign bit of the remainder. When the calculator 22 supplies the final result in a redundant format,

10 block 24 produces the final result in binary format.

FIG. 2 illustrates a more detailed embodiment of divide/square root calculator 22. Calculator 22 uses the SRT Radix-2 algorithm, using redundant quotient digits. While each iteration of

15 the divide generates one bit of the quotient, the divider performs two iterations per cycle, and consequently produces two quotient bits per cycle. In the square root mode, the hardware performs one iteration per cycle, producing one result bit per

20 cycle. The hardware required to perform one iteration is substantially duplicated to produce two quotient bits. Both the quotient and the remainder are kept redundantly within the divide unit and their components are combined subsequently to determine

25 their value.

In the divide mode, divisor D is input through register 100 so that the divisor D and its complement are input to multiplexers 102 and 104. Dividend X is input through register 106 to

multiplexer 108. Partial Remainder-Save (PRS) register 110 and Partial Remainder-Carry (PRC) register 112 store the sum (Ws) and carry (Wc) components of partial remainder (W[j-1]) that were

5   calculated during the prior iteration (j-1). W[j-1] and the output of multiplexer 102 are input to adder 114, which provides the current iteration partial remainder consisting of sum (Ws[j]) and carry (Wc[j]) to selector 116 and to multiplexers 108 and 118.

10   During the next iteration (j+1), multiplexer 104 provides the divisor D and multiplexers 108 and 118 provide the sum (Ws[j]) and carry (Wc[j]) portions of the partial remainder from the prior iteration (j) to adder 120. Multiplexer 108 also supplies the

15   dividend X to adder 120. Adder 120 generates the partial remainder (Ws[j+1] and Wc[j+1]) for iteration (j+1) and supplies these to registers 110 and 112 for use by adder 114 during the next (j+2) iteration.

Selector 116 receives the partial remainder

20   (Ws[j] and Wc[j]) from adder 114 and selector 122 receives the partial remainder (Ws[j+1] and $\bar{Wc}$[j+1]) from adder 120. Selector 116 generates an estimated quotient bit (S_{j+1}) for the next iteration (j+1) based on the four most significant bits of Wc and Ws

25   generated by adder 114, and selector 122 generates an estimated quotient bit (S_{j+2}) for the following iteration (j+2), based on the four most significant bits of Wc and Ws generated by adder 120.

A Radix-R algorithm produces R-1 possible quotient bits 0, 1,..., R-1. Each iteration of the divide requires an exact calculation of the current remainder to determine the next quotient bit. The

5   SRT introduces redundant quotient bits. For instance, in the case of a Radix-2 algorithm, the quotient bits may be -1, 0 and 1. With SRT, the remainder does not need to be known to an exact value for each iteration of the divide. Instead, the

10   remainder need only be known within certain ranges. With a small enough error built into the remainder estimation, the quotient bits can be identified from the range which contains the partial remainder. For Radix-2, in order for the divide to remain bounded,

15   the next partial remainder can not be more than the value of the divisor, nor can it be less than -1*divisor. Therefore, the absolute value of the next partial remainder must be no greater than the divisor: $|W(j+1)| \leq D$.

20   Selectors 116 and 122 examine the four most significant bits Ws and $\overline{Wc}$ to perform the quotient estimation. The following quotient selection logic will keep the partial remainder in bound:

     If $\tilde{}2W[j]$ is the truncated version of

25   $2W[j]$, where $W[j]=Wc[j]+Ws[j]$,

          If $0 \leq \tilde{}2W[j] \leq 3$,   then $S_{j+1} = 1$

          If $\tilde{}2W[j] = -1$,   then $S_{j+1} = 0$

          If $-5 \leq \tilde{}2W[j] \leq -2$,   then $S_{j+1} = -1$

The four bits of ~2W[j] for quotient estimation are the four integer bits, i.e. three integer bits and one fraction bit of ~W[j], represented as Y3 Y2 Y1.Y0

5 The quotient selection logic is based on the sum of the four most significant bits of Ws[j]+Wc[j]:

Magnitude of the quotient: Qm=!(P2&P1&P0)

Sign of the quotient:

$$Qs=P3^{\wedge}(G2\,|\,P2\&G1\,|\,P2\&G1\,|\,P2\&P1\&G0)$$

10 Carry propagation $Pi=Ci^{\wedge}Si$

Carry generation $Gi=Ci\&Si$

Qm	Qs	S_{j+1}
0	x	0
1	0	1
1	1	-1

15 Here, Ci and Si are the carry and sum part of each bit in ~W[j].

Thus, the divide algorithm for dividend X, where 1≤X<2, and divisor D, where 1≤D<2, is:

20 Initially j=0, S_0=1, and W[0]=X-D,

1.   Quotient estimation based on partial remainder W[j]: S_{j+1}=Qsel(W[j])

2.   Calculate next partial remainder:
$W[j+1]=2*W[j]-S_{j+1}*D$

25 3.   Iteratively repeat steps 1 and 2 until all the quotient bits are generated.

Consequently, if the truncated version ~2W[j] is between 0 and 3, inclusive, S[j+1] is +1, if ~2W[j] is -1, S[j+1] is 0, if ~2W[j] is between -5

and -2, inclusive, S[j+1] is -1.   Selectors 116 and 122 provide the estimated quotient bit to registers 128 and 130, which in turn provide outputs to multiplexers 102, 124, 126 and 132.

5   Shift register 128 accumulates a record (QP) of the number of times a positive divisor was subtracted from the current partial remainder, and register 130 accumulates a record (QN) of the number of times a negative divisor was subtracted from the

10   current partial remainder.   Each quotient bit S[j] enters a value into registers 128 and 130 as follows, where $\overline{QN}$ is the one's complement of QN:

Quotient Value	QP	QN	$\overline{QN}$
1	1	0	1
0	0	0	1
-1	0	1	0

Table 1: Quotient bit representation

Multiplexer 132 provides an output from

15   registers 128 and 130 to shift circuit 134, which in turn provides an output to multiplexer 104. Multiplexers 124 and 126 provide outputs to result adder 24 (FIG. 1).   As will become more evident herein, multiplexer 132 and shift circuit 134 are not

20   used in the divide mode, and quotient bit selector 116 is not used in the square-root mode.

The quotient bit S_{j+2} (or, for iteration j, S_j) selected by selector 122 is accumulated in registers 128 and 130 and is supplied to multiplexer

25   102 to generate the $-S_j*D$ part of the recurrence

divide equation. The result is added by adder 114 to the stored sum and carry of the partial remainder for the prior iteration (j-1) to calculate $W[j]=2*W[j-1]-S_j*D$.   Thus, adder 114 adds $2*W[j-1]$ to $-S_j*D$ to

5   produce the partial remainder $W[j]$ in the form of sum and carry for the current iteration. Selector 116 . supplies the estimated quotient bit S_{j+1} to registers 128 and 130 for the next (j+1) iteration. Adder 114 also supplies the sum and carry portions of the

10   partial remainder of the j iteration to adder 120 for the j+1 iteration.

Adder 120 generates the partial remainder for the next iteration, which is transferred to registers 110 and 112 for use by adder 114 during the

15   next (j+2) iteration.   The four most significant digits of the partial remainder from adder 120 are used by selector 122 to provide an estimated quotient bit (S_{j+2}) for the j+2 iteration.

During the first iteration of the divide

20   process (j=0), registers 110, 112, 128 and 130 are empty, so W_0 is zero. Consequently, S_1 selected by selector 116 is 1.   During the second iteration, (j=1) adder 120 receives zero-valued Ws_0 and Ws_1 and generates a partial remainder of -D.   The quotient

25   bit S_2 is selected by selector 122, and the partial remainder and accumulated quotient bits S_0 and S_1 are transferred by multiplexers 124 and 126 to result adder 24 (FIG. 1).   Similarly, during the third and fourth iteration another partial remainder and

quotient bits S_2 and S_3 are generated and transferred to the result adder. Registers 128 and 130 accumulate a record of the number of times a positive divisor was subtracted from a current partial

5   remainder (QP) and the number of times a negative divisor was subtracted from a current partial . remainder (QN).

At the end of the divide operation, multiplexers 124 and 126 are operated first to

10   transfer the accumulated partial remainder from adder 120 to the result adder 24 (FIG. 1) and then to transfer the accumulated quotient components of QP and QN from registers 128 and 130 to result adder 24. Result adder 24 generates the value of the remainder

15   and checks for the sign and zero value. A second partial remainder is generated by computing W[1]+D, where W[1] is the partial remainder after the last iteration 1. The sticky bit is set based on the sign and the value of the two remainders. The sticky bit

20   is an indicator of floating point precision.

The final quotient is calculated in result adder 24 by subtracting QN from QP and adding a possible carry-in value. Since QN is stored in register 130 in 1's complement form as $\overline{QN}$, it can be

25   added to QP. Hence, the quotient is calculated as $QP + \overline{QN} + carry - in$. The carry-in is 1 if the first remainder of the divide is 0 or positive. Otherwise, the quotient must be corrected one smaller by using a carry-in of 0.

For IEEE 754 floating point values, single precision result is 26 bits (24 bits of fraction plus a guard and round bit) with the remainder determining the sticky bit. The double precision result is 55 its (53 bits of fraction plus a guard and round bit) with the remainder determining the sticky bit.

The square root algorithm is similar to the divide algorithm using the same quotient select logic but with a different partial remainder calculation. The requirement for the quotient select is also to guaranty the partial remainder is bounded. The operand range is [0.5, 2), so that if the exponent of the floating point number is an odd number the mantissa may be right-shifted one bit to change the exponent to an even number.

The square root algorithm for a number X, where $0.5 \leq X < 2$ is

Initially $j=0$, $S_0=1$, and $W[0]=X-1$

1. Quotient estimation based on $W[j]$:

$$S_{j+1}=Qsel(W[j])$$

2. Calculate next partial remainder $W[j+1]$:

$$S[j]=\sum_{i=0}^{j} S_1 \bullet 2^{-i}$$

$$W[j+1]=2W[j]-2S[j]S_{j+1}-S_{j+1}^2 \bullet 2^{-(j+1)}$$

3. Iteratively repeat steps 1 and 2 until all quotient bits are generated, where $W[j]$ is the partial remainder at step j, S_{j+1} is the estimated result for step $j+1$, $W[j+1]$

is the partial remainder at step j+1 and

S[j] is the accumulated result to step j.

Both the partial remainder and result are accumulated in a redundant number format as described

5 above. Consequently, the partial remainder calculation of the square root requires two carry-save adders 114 and 120.

Referring to FIG. 2, the partial remainder is generated in selector 122 in redundant form,

10 W[j]=Wc[j]+Ws[j]. Shift registers 128 and 130 will store the generated result bits S[j] to the j-th iteration, $S[j] = QP - QN = QP + \overline{QN}$. Substituting this value of S[j] into the square root remainder equation, $W[j+1] = 2W[j] - 2S[j]S_{j+1} - S_{j+1}^2 \bullet 2^{-(j+1)}$, when S_{j+1} is

15 0, W[j+1]=2Wc[j]+2Ws[j], where Wc[j] and Ws[j] are the carry and sum parts of the partial remainder in the j-th iteration. In this case, adders 114 and 120 are only adding a zero to the partial remainder.

When S_{j+1} is 1, -2S[j]S_{j+1} is 2QN[j], and

20 $S_{j+1}^2 \bullet 2^{-(j+1)}$ is $-[2QP[j] + 2^{-(j+1)}]$, so

$$W[j+1] = 2Wc[j] + 2Ws[j] + 2QN[j] - [2QP[j] + 2^{-(j+1)}].$$

In this case, multiplexer 102 supplies 2QN[j] to adder 114, which in turns adds $2Wc[j] + 2Ws[j] + 2QN[j]$, producing the result in the sum and carry format.

25 $-[2QP[j] + 2^{-(j+1)}]$ is computed by shifter 134 and is supplied to adder 120 by multiplexer 104. Adder 120 adds $-[2QP[j] + 2^{-(j+1)}]$ to the result from adder 114 to derive W[j+1].

When S_{j+1} is -1, -2S[j]S_{j+1} is 2PN[j], and $S_{j+1}^2 \bullet 2^{-(j+1)}$ is $-[2QN[j]+2^{-(j+1)}]$, so

$$W[j+1] = 2Wc[j] + 2Ws[j] + 2QP[j] - [2QN[j] + 2^{-(j+1)}]$$

In this case, multiplexer 102 supplies 2QP[j] to
5   adder 114, which in turns adds $2Wc[j]+2Ws[j]+2QP[j]$,
producing the result in the sum and carry format.
$-[2QN[j]+2^{-(j+1)}]$ is computed by shifter 134 and is
supplied to adder 120 by multiplexer 104. Adder 120
adds $-[2QN[j]+2^{-(j+1)}]$ to the result from adder 114 to
10 derive W[j+1].

Shifter circuit 134 shifts the input '1'
one bit to the left and forces the -(j+1) bit
position to binary '1'. The one bit shift
effectively provides -2QP[j] where S_{j+1} is '1' or
15 -2QN[j] where S_{j+1} is '-1'. The force of a '1' at the
-(j+1) bit position effectively adds $-2^{-(j+1)}$ to the
shifted result. For example, if the output from
multiplexer 132 (representing QP or QN) is a ten-bit
bus, and the result S[j] to the fifth iteration is
20 '0101000000', at the fifth iteration (j=5) shifter
134 shifts the partial remainder left by one-bit to
'1010000000', and forces a '1' at the -6 bit position
-(j+1), resulting in the mantissa of the partial
remainder of '1010010000'. The result is summed with
25 Ws[j] and Wc[j] by adder 120 to derive Ws[j+1] and
Wc[j+1] which are stored in registers 110 and 112 for
use as Ws[j] and Wc[j], respectively, during the next
iteration.

Selector 122 selects the estimated result S_{j+1}, which is inserted into registers 128 and 130 as QP and $\overline{QN}$ for the next iteration. The result of selector 116 is not used for square root function.

5 Since both adders 114 and 120 are used to compute one iteration of square root, only one result bit is generated each cycle through the circuit. The result bit is estimated using the same logic as described in the divider section. The result is maintained in

10 identical fashion and shifted into shift registers 128 and 130, one bit at a time.

After all the iterations are completed, multiplexers 124 and 126 are operated first to transfer the accumulated partial remainder from adder

15 120 to the result adder 24 (FIG. 1) and then to transfer the accumulated values of QP and QN from registers 128 and 130 to result adder 24. The result adder generates the value of the remainder and checks for the sign and zero value. A second partial

20 remainder is generated by computing $W[l] + (2)S - 2^{-(l+1)}$, where S is the accumulated result to the last iteration and W[1] is the partial remainder after the last iteration 1. The sticky bit is set based on the sign and the value of the two remainders.

25 The accumulated result components are transferred after transfer of the accumulated partial remainder. The final result is calculated in result adder 24 by subtracting QN from QP as: $QP + \overline{QN} + carry - in$. The carry-in is 1 if the first

remainder of the divide is 0 or positive. Otherwise, the result must be corrected one smaller by using a carry-in of 0.

For IEEE 754 floating point values, single precision result is 26 bits (24 bits of fraction plus a guard and round bit) with the remainder determining the sticky bit. The double precision result is 55 bits (53 bits of fraction plus a guard and round bit) with the remainder determining the sticky bit.

In the case of the first iteration if the initial partial remainder W[0] is equal to X-1 where X is the operand, the initial partial remainder W[0] is calculated by adder 120 while adder 114 is bypassed.

The mantissa calculator 22 (FIG. 2) calculates the mantissa to transfer the partial remainder from adder 120. The accumulated quotient or result bits are then transferred from registers 128 and 130. The result bits are accumulated for operation on adder 120, and the partial remainder is transferred to result adder 24. The partial remainder is employed by result adder 24 to generate the resulting mantissa and establish the final value of the exponent in accordance with traditional divide and square root algorithms.

The present invention thus provides an improved floating point divide and square root processor in which the square root algorithm is mapped to the divide algorithm hardware over plural

divide operations. Hence, during each cycle, the processor provides plural quotient bits in the divide mode and a single result bit in the square root mode. The divide and square root algorithms share a common

5 result prediction logic. The final corrective step in the square root result is performed based on the partial remainder.

The processor is small in size for reasonable performance and may be used for high

10 frequency operation. While the invention is described herein as employing a Radix-2 algorithm, high performance may be achieved by duplicating the hardware or employing a higher radix algorithm. Nevertheless, use of Radix-2 algorithm permits

15 simpler testing of the divide and square root processor, compared to multiplication-based and Radix-4-based processors.

Although the present invention has been described with reference to preferred embodiments,

20 workers skilled in the art will recognize that changes may be made in form and detail without departing from the spirit and scope of the invention.